Prova Final de Linguagens de Programação
- DCC024 -
Ciência da Computação

Ao concordar participar da prova, você dá **sua palavra de honra** que suas respostas são fruto único do seu trabalho. Você pode consultar a internet, por exemplo, mas não pode consultar outros seres vivos para fazer a prova.

As regras do jogo:

- Você pode consultar entidades inanimadas, mas não pode consultar entidades animadas.
- Sua solução deve estar distribuída em três arquivos:
  - `sol.sml`: solução da questão 1.
  - `sol.py`: solução da questão 2.
  - `sol.pl`: solução da questão 3.
- Coloque os arquivos em uma pasta com o seu nome (`\fulano_de_tal\`), e compacte-os em um arquivo `sol.zip` (ou `sol.gz`). Envie `sol.zip` para `fernando@dcc.ufmg.br` até as 9h10 do dia 19 de Junho.
- Seu e-mail deve ter o título `Prova_DCC024`. Não use qualquer outro título.
- Escreva seu nome completo no corpo do e-mail, e, possivelmente, a resposta da questão extra. Não escreva o código dos arquivos no e-mail: envie `sol.zip` como anexo.
- Caso você queira reenviar suas respostas, simplesmente responda a mensagem de seu e-mail anterior. Não envie e-mails separados! Respostas não são corrigidas após 9h10.
- Não há como tirar dúvidas durante a prova. Lembre-se da cláusula sobre entidades animadas. Isso inclui escrever e-mails para o instrutor.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Extra 0.5

Questão extra (0.5): Responda a pergunta de Schtroumpf.

1. (10 Pontos) Nesta questão, você deverá implementar uma função `mywhl`, em SML. Essa função simula a funcionalidade do comando `while` da linguagem C, e possui o seguinte tipo: `int -> 'a -> ('a -> 'a) -> 'a`. Ela recebe um contador `n`, um acumulador `acc` e uma função `cmd`. Ela então aplica `cmd` sobre `acc` um número de vezes igual a `n`:

   ```
   mywhl n acc cmd = cmd (cmd (...  cmd acc))
   ```

   Alguns exemplos da execução da função `mywhl` podem ser vistos logo abaixo:

   ```
   - mywhl 3 1 (fn x => x + 1);
   val it = 4 : int
   - mywhl 7 3 (fn x => x + 1);
   val it = 10 : int
   - mywhl 3 2 (fn x => x * x);
   val it = 256 : int
   - mywhl 3 nil (fn x => "dcc024" :: x) ;
   val it = ["dcc024","dcc024","dcc024"] : string list
   - mywhl 4 "" (fn x => "+" ^ x ^ "-");
   val it = "++++----" : string
   - mywhl 4 "dcc024" (fn x => "(" ^ x ^ ")");
   val it = "((((dcc024))))" : string
   ```

   (a) (4 Pontos) Implemente a função `mywhl`.

   (b) (3 Pontos) Utilize a função `mywhl` para implementar uma função `nthPower`, de tipo `int -> int`. Essa função recebe um número n, e retorna n^n. Por exemplo[1]:

   ```
   - nthPower 1;
   val it = 1 : int
   - nthPower 2;
   val it = 4 : int
   - map nthPower [1, 2, 3, 4, 5, 6, 7];
   val it = [1,4,27,256,3125,46656,823543] : int list
   ```

   (c) (3 Pontos) Utilize a função `mywhl` para implementar uma função `fact`, de tipo `int -> int`. Essa função retorna o fatorial de um número. O fatorial de um número negativo é 1. Por exemplo:

   ```
   - fact 0;
   val it = 1 : int
   - fact 2;
   val it = 2 : int
   - fact ~1;
   val it = 1 : int
   - map fact [~5, ~4, ~3, ~2, ~1, 0, 1, 2, 3, 4, 5];
   val it = [1,1,1,1,1,1,1,2,6,24,120] : int list
   ```

[1] Note que é fácil implementar essa função (e também a próxima), diretamente em SML, sem usar `mywhl`. Mas, nesta questão, você deve usar `mywhl`, e somente `mywhl`, para produzir qualquer tipo de iteração!

2. Esta questão refere-se à manipulação de *strings* em Python. O termo *substring* é definido como uma fatia de uma string. Em outras palavras, dada uma string `s`, uma substring de `s` é uma sequência *contígua* dos caracteres de `s`.

   (a) (4 Pontos) Escreva uma função `largestIncreasingSubstr(s)` que produza uma tabela `L` com a seguinte propriedade: `L[i]` é a maior substring *crescente* a partir da posição `i` da string `s`. Por exemplo:

```
>>> largests = largestIncreasingSubstr("Domingos")

>>> for s in largests:                    >>> for i in range(len(largests))
...   print(s)                            ...   print (i, largests[i])
...                                       ...
Do                                        (0, 'Do')
o                                         (1, 'o')
m                                         (2, 'm')
in                                        (3, 'in')
n                                         (4, 'n')
gos                                       (5, 'gos')
os                                        (6, 'os')
s                                         (7, 's')
```

   (b) (3 Pontos) Crie uma função `findLargest(s)`, que retorne a maior substring da string `s` que possua todos os caracteres em ordem lexicográfica crescente. Por exemplo:

```
>>> print("Largest string: <" + findLargest("abcabcdababcdeab")) + ">"
Largest string: <abcde>
>>> print("Largest string: <" + findLargest("abcabcdababcdab")) + ">"
Largest string: <abcd>
>>> print("Largest string: <" + findLargest("xbcxbcdxbxbcdxb")) + ">"
Largest string: <bcdx>
>>> print("Largest string: <" + findLargest("edcba")) + ">"
Largest string: <a>
>>> print("Largest string: <" + findLargest("edxcba")) + ">"
Largest string: <dx>
>>> print("Largest string: <" + findLargest("")) + ">"
Largest string: <>
```

   (c) (3 Pontos) Crie uma função `findLargestThan(s, K)`, que retorne todas as substrings de `s`, constituídas por caracteres em ordem lexicográfica crescente, que possuam mais que `K` caracteres. Por exemplo:

```
>>> findLargestThan("ababcababcdab", 2)
['abc', 'abcd', 'bcd']
>>> findLargestThan("abcdefgh", 3)
['abcdefgh', 'bcdefgh', 'cdefgh', 'defgh', 'efgh']
>>> findLargestThan("edcba", 3)
[]
>>> findLargestThan("edcba", 0)
['e', 'd', 'c', 'b', 'a']
>>> findLargestThan("", 2)
[]
```

3. Nesta questão iremos descrever um jogo de palavras cruzadas em Prolog. Por exemplo, o tabuleiro abaixo é descrito por 36 predicados[2]:

Predicates

char(0, 0, s).	char(1, 0, o).	char(2, 0, m).	char(3, 0, a).	char(4, 0, r).	char(5, 0, l).
char(0, 1, a).	char(1, 1, c).	char(2, 1, o).	char(3, 1, c).	char(4, 1, o).	char(5, 1, i).
char(0, 2, l).	char(1, 2, a).	char(2, 2, t).	char(3, 2, a).	char(4, 2, l).	char(5, 2, r).
char(0, 3, t).	char(1, 3, r).	char(2, 3, i).	char(3, 3, l).	char(4, 3, h).	char(5, 3, a).
char(0, 4, a).	char(1, 4, o).	char(2, 4, m).	char(3, 4, d).	char(4, 4, a).	char(5, 4, m).
char(0, 5, r).	char(1, 5, e).	char(2, 5, a).	char(3, 5, o).	char(4, 5, v).	char(5, 5, o).

Palavra Cruzada

S	A	L	T	A	R
O	C	A	R	O	E
M	O	T	I	M	A
A	C	A	L	D	O
R	O	L	H	A	V
L	I	R	A	M	O

(a) (3 Pontos) Crie um predicado `hsearch(X, Y, L)`, que seja verdadeiro se `L` for uma lista que descreva uma palavra começando na posição (`X`, `Y`) da matriz de palavra cruzada. Essa palavra deve ser lida *horizontalmente*, da esquerda para a direita. Por exemplo, considerando-se a matriz vista acima, temos:

```
?- char(1, 1, X).
X = c ;
false.
?- hsearch(1, 1, [c]).
true ;
false.
?- hsearch(1, 1, [c, a, r, o]).
true ;
false.
?- hsearch(1, 1, [c, a, r, r, o]).
false.
```

(b) (3 Pontos) Crie um predicado `vsearch(X, Y, L)`, que seja verdadeiro se `L` for uma lista que descreva uma palavra começando na posição (`X`, `Y`) da matriz de palavra cruzada. Essa palavra deve ser lida *verticalmente*, de cima para baixo. Por exemplo, considerando-se a matriz vista acima, temos:

```
?- vsearch(1, 1, [c, o, c, o]).
true ;
false.
?- vsearch(1, 1, [t, r, i, l, h, a]).
false.
?- vsearch(0, 3, [t, r, i, l, h, a]).
true ;
false.
```

(c) (2 Pontos) Crie um predicado `wmember(W)`, que seja verdadeiro caso `W` for uma lista contendo uma palavra presente na matriz de palavras cruzadas. Palavras deve ser lidas horizontalmente (sempre da esquerda para a direita), ou verticalmente (sempre de cima para baixo). Note que você pode reusar `hsearch` e `vsearch`, os predicados vistos na questão acima. Por exemplo:

[2]Não assuma que a matriz de palavras cruzadas é fixa. Seu programa será testado com uma matriz diferente. Mas, se quiser um caso de testes, fique à vontade para usar esses predicados: `char(0, 0, s).` `char(0, 1, a).` `char(0, 2, l).` `char(0, 3, t).` `char(0, 4, a).` `char(0, 5, r).` `char(1, 0, o).` `char(1, 1, c).` `char(1, 2, a).` `char(1, 3, r).` `char(1, 4, o).` `char(1, 5, e).` `char(2, 0, m).` `char(2, 1, o).` `char(2, 2, t).` `char(2, 3, i).` `char(2, 4, m).` `char(2, 5, a).` `char(3, 0, a).` `char(3, 1, c).` `char(3, 2, a).` `char(3, 3, l).` `char(3, 4, d).` `char(3, 5, o).` `char(4, 0, r).` `char(4, 1, o).` `char(4, 2, l).` `char(4, 3, h).` `char(4, 4, a).` `char(4, 5, v).` `char(5, 0, l).` `char(5, 1, i).` `char(5, 2, r).` `char(5, 3, a).` `char(5, 4, m).` `char(5, 5, o).`

```
?- wmember([s, o, m, a]).
true ;
false.
?- wmember([s, o, m, a, s]).
false.
?- wmember([c, a, l, d, o]).
true ;
false.
?- wmember([o, X, a]).
X = c ;
X = m ;
false.
```

(d) (2 Pontos) Crie um predicado `answers(L, S)`, que seja verdadeiro caso `L` seja uma lista de listas de letras, e `S` seja uma sublista de `L`, formada somente pelas palavras que estão presentes na matriz de palavras cruzadas. Por exemplo:

```
?- answers([[s, o, m, a, r], [c, o, c, o], [c, a, r, r, o], [c, a, r, o]], S).
S = [[s, o, m, a, r], [c, o, c, o], [c, a, r, o]].
?- answers(nil, S).
S = [].
?- answers([[c, a, c], [c, o, c, a], [a, r, o], [c, a, r, o]], S).
S = [[a, r, o], [c, a, r, o]].
?- answers([[X1, X2, X3, X4, X5, X6]], S), length(S, N).
S = [[s, a, l, t, a, r], [o, c, a, r, o, e], [m, o, t, i, m, a],
     [a, c, a, l, d|...], [r, o, l, h|...], [l, i, r|...],
     [s, o|...], [a|...], [...|...]|...],
N = 12.
```